The Iterator Pattern

Gerald Britton
Pluralsight Author

@GeraldBritton www.linkedin.com/in/geraldbritton

Collections

Iteration

Developer creativity

Hide the implementation

Iterator pattern

Overview

Employee collection

Holds employee objects

Clients iterate over the collection

Collection exposes method for iteration

Hide collection implementation

Many ways to do that

No conformity

Demo

Collection of employees

Could be a list, set, dictionary, tree

One possibility for iterating over it

Iterator

Classification: Behavioral

Adds new abilities to a collection

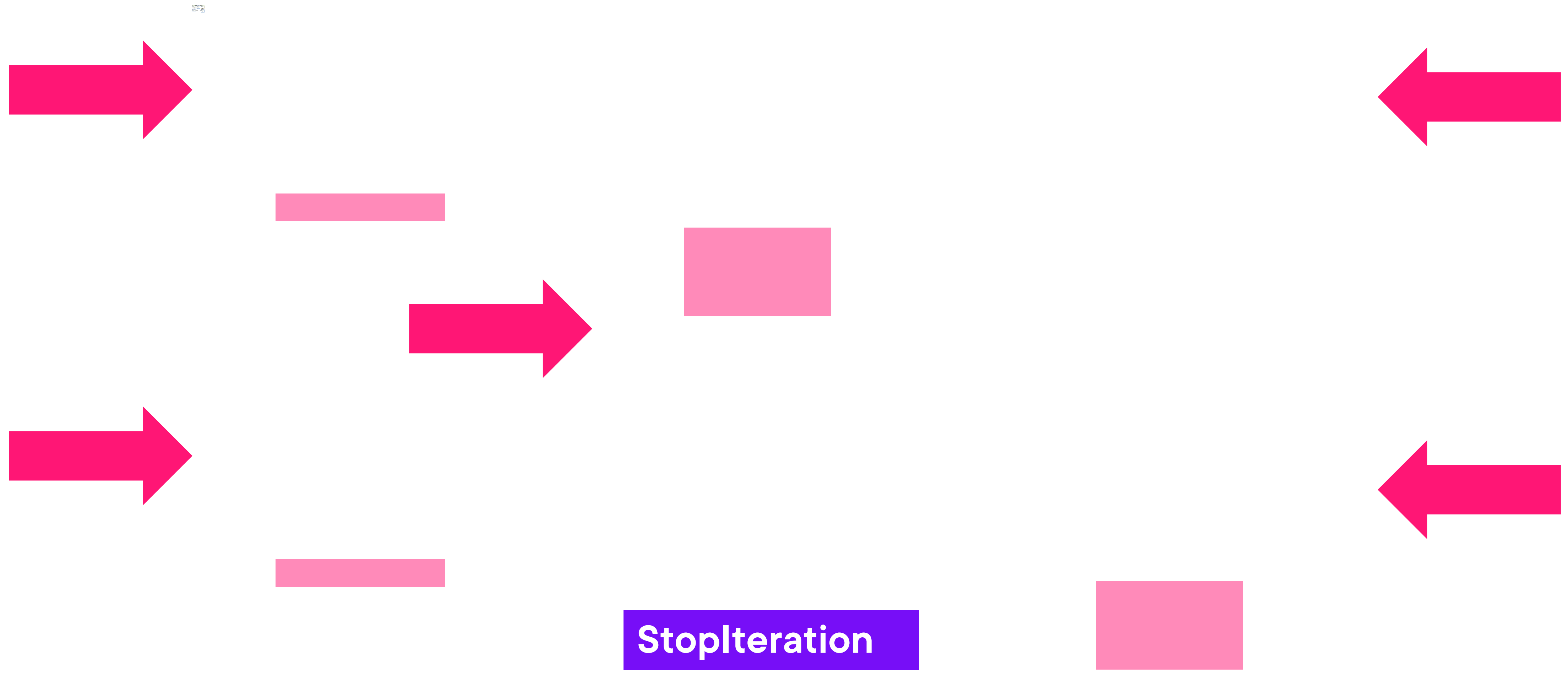
Iterate over the elements

Without exposing the underlying representation

- Preserves encapsulation

Also known as the Cursor Pattern

Iterator Pattern Structure



Python Iterators

Two different iterator objects

Sequence iterator

- `__getitem__()`

Callable object

- `__iter__()` and `__next__()`
- `next()` in Python 2.x

Built into the compiler

Collections module

- Iterable and Iterator
- Sequence

Demo

Build iterators for the collections

Look at both types

Iterable = Iterator

Use them in the main program

Complete the `print_summary` function

Demo

Use generator expressions

- `(x for x in iterable)`
- `(f(x) for x in iterable)`
- `(f(x) for x in iterable if <condition>)`

Core Python on Pluralsight.com

Consequences

Simple, standard interface

Collection implementation can vary

- n-way tree: depth or breadth first

Multiple active, independent iterators

Python generators make it easy!

Summary

When to use Iterator?

Iterate over a collection

Preserve encapsulation

Multiple active iterations

Uniform interface